

EXPRESS MAIL NO. EL 743 811 552 US

Rockwell Ref: 01SW155

Docket No. 110003.97656

PATENT APPLICATION FOR

INDUSTRIAL CONTROLLER INTERFACE
PROVIDING STANDARDIZED OBJECT ACCESS

by

Jayne Edwards
David J. Lillie

00067206 092801

INDUSTRIAL CONTROLLER INTERFACE
PROVIDING STANDARDIZED OBJECT ACCESS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation-in-part of a U.S. application Serial No. 09/904,929 entitled: "Internet Object Interface for Industrial Controller".

STATEMENT REGARDING FEDERALLY
SPONSORED RESEARCH OR DEVELOPMENT

[0002] --

BACKGROUND OF THE INVENTION

[0003] The present invention relates to a computer program providing an interface for an industrial controller, and in particular, to a program providing a uniform interface to different software objects, possibly from different vendors.

[0004] Industrial controllers are special purpose computers used for the control of industrial processes and the like. While executing a stored control program, they read inputs from the control process and, for the logic of the control program, provide outputs to the control process. Such controllers differ from conventional computers in two major respects.

[0005] First, the architecture of the industrial controller is designed to allow it to be easily reconfigured to meet the demands of the particular process being controlled. Normally, therefore, the industrial controller is in modular form. The user selects different modules to be assembled together in a rack and linked together on a back plane in the rack. Varying numbers of inputs and outputs are accommodated with input/output (I/O) modules that may also fit in the rack or in a remote rack connected to the first rack over a control network.

[0006] The ability to modify the industrial controller to match the control process is furthered by writing custom program to be executed on the industrial controller. These programs may be both for the control of the industrial process and to provide human machine interfaces (HMIs), the latter which often depict a graphic representation of all or part of the process animated to show its current operating state. The use of custom

result in a functionally limited set of objects. While these objects may be supplemented with objects from additional vendors, switching between object protocols of different vendors can be cumbersome. Further, using any proprietary object set carries with it the risk that the objects will be modified by the vendor or that the vendor will cease supporting particular objects or all objects. This lack of stability in the programming environment is undesirable in the area of industrial control where considerable cost is invested in the custom controller program.

BRIEF SUMMARY OF THE PREFERRED EMBODIMENT

[0011] The present invention provides a "standard object protocol" overlying a set of different software object types. A set of specially prepared "object providers" link an arbitrary set of proprietary software objects with a client program using this standard object protocol. Each object provider translates between the standard object protocol and the proprietary protocol used by an associated one of the proprietary software objects. In this way, the invention makes the peculiarities of object protocols of proprietary objects invisible to the programmer, allowing software objects to be mixed from different vendors and to be changed in the future without unduly affecting the client program that uses them. By freeing the programmer from proprietary object protocols, improved stability, uniformity, and functional range of the objects is provided.

[0012] Specifically, the present invention provides an object-based interface for an industrial control system. In the invention, the interface receives communications from a client program employing a standard object protocol. The interface provides a set of software objects including at least two proprietary objects having differing proprietary object protocols that also differ from the standard object protocol. At least two object providers communicate with the two proprietary objects, one with each, to translate between standard object protocol and the protocol used by the proprietary objects.

[0013] It is thus one object of the invention to provide a standard object protocol that may span and unify a programmer's use of third party or proprietary software objects, each of which may have a unique and differing protocol.

[0014] It is another object of the invention to allow flexibility in changing the underlying software objects as competition and innovation provide improved and new software object models.

[0015] The standardized and proprietary object protocols may control object features such as object creation, object destruction, setting parameters of the objects, and invoking methods of the objects.

[0016] Thus it is another object of the invention to extract the essential features of a variety of objects, such features as may be used to provide a framework for translation between arbitrary object protocols and a standard object protocol.

[0017] The standard object protocol may include discover instructions and the object providers may respond to the discovery instructions by identifying object features of the software objects with which they are associated. These object features may include the parameters of the objects and the methods of the objects.

[0018] Thus, it is another object of the invention to provide for a standard object protocol that accommodates variations between objects by allowing dynamic discovery of object features.

[0019] A client program may communicate with the server program over a network, and the object providers may be associated with network addresses.

[0020] Thus, it is another object of the invention to provide a system well adapted to remote operation over the Internet or the like.

[0021] The object providers may be software objects that provide encapsulation of data passed to the proprietary software objects.

[0022] Thus, it is another object of the invention to provide an interface to proprietary software objects that itself provides the benefits of the object model.

[0023] The foregoing objects and advantages may not apply to all embodiments of the invention and are not intended to define the scope of the invention, for which purpose claims are provided. In the following description, reference is made to the accompanying drawings, which form a part hereof, and in which there is shown by way of illustration, a preferred embodiment of the invention. Such embodiment also does not define the scope of the invention and reference must be made therefore to the claims for this purpose.

BRIEF DESCRIPTION OF THE DRAWINGS

[0024] Fig. 1 is a simplified block diagram of an Internet connection between a remote browser and two servers, the first which includes an HMI applet and the second which provides an interface to an industrial controller;

- [0025] Fig. 2 is a flow chart showing the connection of the browser to the industrial controller for the implementation of an HMI;
- [0026] Fig. 3 is a detail block diagram of the software components of the browser and server showing structures within the server for implementing an object-based interface protocol and their connection to an external industrial controller;
- [0027] Fig. 4 is a graphic representation of one of the objects of the server of Fig. 3 showing its properties, methods and events;
- [0028] Fig. 5 is a simplified example of an industrial process including a conveyor belt showing an output object for actuating the conveyor belt in an input object for measuring passage of objects on the conveyor belt;
- [0029] Fig. 6 is a graphical representation similar to that of Fig. 4 showing construction of an object making use of the input and output objects of Fig. 5;
- [0030] Fig. 7 is a flow chart showing transmission of instructions from the remote terminal to the server;
- [0031] Fig. 8 is a graphical display object that may be implemented by the present invention;
- [0032] Fig. 9 is a fragmentary block diagram similar to Fig. 3, showing an interface of the present invention implemented by the server to provide for a standard object protocol including object providers forming translating between the standard object protocol and proprietary objects;
- [0033] Fig. 10 is a partial, detailed view of Fig. 9 showing object providers as plugged into the server framework as relating to particular proprietary objects and showing an object communication message adhering to the standard object protocol, and a instanced object formed by the combination of a proprietary object and an object provider; and
- [0034] Fig. 11 is a graphical representation of an object provider of Fig. 10 together with an associated proprietary object showing its linkage to various classes and parameter files.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

System Overview

[0035] Referring now to Fig. 1, the present invention will be described in the context of a human machine interface (HMI) implementable on a browser-enabled, remote computer 10. Generally, the present invention will be applicable to other remote operations requiring an interface to an industrial control system over the Internet.

can be to a different Web page on the same Web server 16, however, this does not provide the benefit of centralized distribution of the applet program 30 as will become apparent from the following description. Executing the applet program 30 as indicated by process block 34 of Fig. 2 causes the browser to exchange data with the Web server 16 using the SOAP protocol for the exchange of object-oriented instructions within the encapsulation of a SOAP envelope 35.

Object Interface

[0047] Referring now to Fig. 4, the server-side HMI program 37 provides a standard set of preconfigured I/O objects that encapsulate physical inputs and outputs 39 of the industrial controller 14. For example, output object 38' may provide a binary output corresponding to an output 39 of the industrial controller 14 such as may control a particular device in the industrial process 41. Per standard object convention, the object 38' provides a number of properties 66 that represent variables affecting the state of the object 38'. In this example of a binary output object 38', only a single property indicating either that the output is ON (i.e., TRUE or HIGH) or OFF (i.e., FALSE or LOW) is provided. As described through the agency of the communications program 64 (shown in Fig. 3) this state property 66 may be written to or read to reveal or change the corresponding output 39 associated with the industrial controller 14.

[0048] The object 38' also includes a method 68 invoking code internal to the object 38' for toggling the output state property in this case that is changing the state from ON to OFF or from OFF to ON depending on its previous state.

[0049] The output object 38 may also provide for events being signals provided by the object 38' under certain conditions, in this case, a state change event 70.

[0050] In a preferred embodiment, each input and output 39 of the industrial controller 14 will have an I/O object 38 assigned to it; however, other abstract objects 38 may also be created. As will be described, these abstract objects 38 may be constructed from the remote computer 10 and may have the properties of persistence, that is, remaining even after the remote computer 10 has logged off.

[0051] Referring now to Fig. 5, in a simple controlled process 41, a conveyor belt 72 may be operated by a motor 74 receiving a signal 76 from an output 80 of the industrial controller 14. That output 80 may have a corresponding output object 38a as shown in Fig. 6. A photoelectric sensor 82 may be positioned over the top of the conveyor belt 72 to detect the passage of parts 84 across the conveyor belt 72. A photoelectric sensor may

by the objects. By providing for persistence of the constructed objects at the server side, complete flexibility of remote computer location and type is still preserved, making it possible to call up a particular HMI for an arbitrary industrial controller 14 from any browser enabled remote computer 10 anywhere in the world.

Object Interface Instructions

[0055] The ability to create a desired HMI through the applet program 30 for an arbitrary industrial controller 14 requires that the I/O objects 38 and other standard abstract objects 38 be discoverable remotely. Accordingly, the present invention provides a special set of object-oriented instructions that allow for discovery of the objects 38 at a particular Web server 16 along with their properties, methods, events, and other characteristics as follows:

Table I Object Discovery Instructions

INSTRUCTION	EXPLANATION
GetEndpoints	Reply to the remote computer 10 with a list of URIs 43 available at the Web server 16.
GetObjects	Reply to the remote computer 10 with a list of objects 38 available at the Web server 16 on a given Endpoint.
GetConstructors	Reply to the remote computer 10 with a list of all constructors for creating new objects available at the Web server 16.
GetProperties	Reply to the remote computer 10 for a particular object with a list of all of the object's properties.
GetActions	Reply to the remote computer 10 for a particular object with a list of all of the object's methods.
GetSubscription	Reply to the remote computer 10 for a particular object with a list of all of the object's events.

[0056] The discovery instructions do not change with changes in the industrial process 41 (for example I/O) or even in upgrades of the applet and thus can be standardized at each Web server 16. On the other hand, the values returned by the discovery instructions change depending both on the industrial process 41 and its control, principally in the form of what objects 38 are available, allowing the precise configuration of the HMI to vary considerably under identical applet programs 30. The implementation of the object

discover instructions is performed by the DISCOVER object interface programs 36 which reviews a table holding object definition as is created for the I/O objects during configuration of the industrial controller 14 and for abstract objects by the HMI program designers.

[0057] The flexibility of the HMI using the applet program requires persistence of the objects generated (so that the objects called by the HMI do not need to be re-instantiated for each session) so the present invention also provides a set of persistence instructions that provide for persistence of an object created by a remote user even after the user has logged off.

Table II Object Persistence Instructions

GetLoadParameters	Reply to the remote computer 10 with a file name for a file holding the parameters for an identified object that must be loaded for the object to be resolved into a persisted state.
GetSaveParameters	Reply to the remote computer 10 with a file name for a file holding the parameters for an identified object that must be saved for the object to be stored into a persisted state.
Load Object	Loads a previously instantiated and saved object.
Save Object	Saving a previously instantiated object using the GetSaveParameters returned file name.

[0058] The Object persistence instructions are implements by the PERSISTENCE object interface program 36 which saves and loads the necessary instantiation data from a non volatile memory portion forming the persistent storage area 42 and which records the file name according to well known file storage techniques.

[0059] The present invention contemplates a complete customization of the HMI by the remote user and thus provides for the construction of objects necessary to create that

handler 81 being part of the server side HMI program which accepts messages denoting events from enables subscriptions into an event queue and processes messages to the remote computers accordingly.

[0063] The objects may in this manner send messages to the applet program over the Internet, which provide part of an effective application programmer's interface or API.

[0064] Referring now to Figs. 3 and 7, a typical interaction between the browser 50 on remote computer 10 and Web server 16 will start by the reception of one or a set of object-oriented instruction from the applet program 30 as initiated by the user to be transmitted by the applet program 30 as indicated by process block 83.

[0065] The applet program 30 as indicated by process block 85 then encapsulates the object-oriented instruction in SOAP protocol for transmission through the browser 50 according to standard techniques used in browser operating system interaction. The browser 50 as indicated by process block 87 transmits this SOAP envelope 35 to the URL of the Web server 16 where it is received as indicated by process block 89 and the SOAP encapsulation removed by the server software 60.

[0066] The underlying object-oriented instructions are passed to the parser 62 as indicated by process block 90 and then passed to the server-side HMI program 37 for processing as has been described.

Standard Object Protocol

[0067] Referring now to Fig. 9, the objects 38 provided by the server side HMI program 37 as described above may be implemented using a variety of third party or proprietary software objects 100, for example, including objects from standard programming languages like Visual Basic, Java or C++ or from operating systems including COM objects, or from other commercial programs that employ objects including graphics programs, and database programs, as well as XML Documents that the invention exposes as objects. In this respect, the term proprietary is intended to indicate only that the protocol is not universally adopted or may be subject to restrictions on use.

[0068] Messages from the parser 62 may be routed through an interceptor 102 which may execute a predetermined set of programs based on receipt of the messages, for example, verifying a license or tallying a billing required for a license fee for use of the object if one is required or providing certain security features or the like. The parser 62 then routes the messages to an object provider group 104 via an endpoint decoder 106 as will be described. The object provider group 104 may respond to the message 105 and

communicate with an object assenter 108 which in turn may communicate with the network 12.

[0069] The object provider group 104 holds object providers 110 that generally provide an interface between a standard object protocol that will be employed in the messages 105 encapsulated in the SOAP envelope 35 and a proprietary software object 100. The object providers 110 are preferably also software objects.

[0070] Referring now to Fig. 10, a message 105 using the standard object protocol delivered via the SOAP protocol 35 for access to the object 38 is received by the object provider group 104 and may communicate with a registered object providers 110 or with an instanced object 38 formed of the combination of an instanced object provider 110 and an instanced proprietary object 100.

[0071] The former case, communication with the uninstanced object provider 110 is used for messages 105 requiring discovery or construction of an object. The object provider 110 is written to respond to the discovery instructions related to the particular object 100 for which it is intended and includes constructor methods.

[0072] In the latter case, the instanced form of the object provider 110, which forms part of object 38, communicates the message to the instanced form of the proprietary objects 100. By encapsulating the communication between the message 105 and the proprietary objects 100 in the structure of the object provider 110, the reliability of the interface process may be ensured. An instance of the object provider 110 and an instance of a software proprietary object 100 create an object 38

[0073] Generally proprietary objects 100 may include objects 100' and 100'' from different vendors and having different protocols. When instanced, the object provider 110 maps methods, events and parameters of the objects 100 and 100' and the protocols for accessing these methods and parameters and subscribing (or canceling a subscription) to these events, to methods and parameters and protocols and events of the standard object protocol of message 105. Where parameters are different but functionally transformable to be equivalent, the object provider 110 provides the functional transformation. Where parameters of the objects 100 may be combinations of other parameters, that combination too may be affected by the object provider 110. The object provider 110 may also provide for default parameters and for default message states and affect a change in naming convention and syntax to provide an abstraction of the underlying proprietary software object interface and thus give the programmer a unified

and standard programming environment. Some methods not realized by the objects 100 can be implemented by the object providers 110.

[0074] In this way, the programmer is insulated from differences in the protocols of the objects 100' and 100".

[0075] Referring to Figs. 10 and 11, a number of steps are required before objects 38 being available to the programmer. First, the uninstanced object provider 110 must be registered with the endpoint decoder 106 so that its existence may be revealed to the programmer via the discovery instructions of messages 105. This is done by providing a tag name (e.g., javaObject) that will be used to identify the object provider 110 and linking it to the name of a class (e.g., JavaObjectProvider) that implements the object.

This may be done using HTML in the present invention with the following instructions.

```
<rip:object-provider-name = "javaObject"
  className="com.rockwell.integration.providers.object.JavaObjectProvider
">
```

[0076] These instructions link the tag name of an object provider 110, "javaObject" to a program 114 that implements the object provider 110 and is represented by arrow 112 in Fig. 11. The tag name is revealed in the discover instruction. The program 114 provides the methods that implement the discovery, persistence, constructors, and access features described above for a particular legacy proprietary object 100.

[0077] After the object provider 110 has been linked to its underlying program, it must be linked to the particular proprietary object 100. This may be done by the following instructions.

```
<rip:object name ="MyWebObject" >
  <rip:object-provider-ref = "javaObject">
    <className>
      com.mycompany.mypackage.MyObject
    <\className>
```

[0078] These instructions simply make the connection between the object provider 110 "javaObject" and the proprietary object 100 "MyObject" and provide tag name "MyWebObject" for the proprietary object 100. This allows the proprietary object 100 to be invoked by the object provider 110 and communication between the two to implement the interface when they are instanced. This linkage is indicated by arrows 113 in Fig. 11.

[0079] In the next step, the object provider 110 exposes its associated object 100 at a particular endpoint on the network such as a URL by which messages 105 may be received. This may be done by the following instructions.

```
<rip:endpoint uri=\mywebobject" default="MyWebObject">  
  <rip:object ref="MyWebObject"/  
</rip:endpoint>
```

[0080] The endpoint connection is used by endpoint decoder 106 described above with respect to Fig. 9 and is represented by arrow 116 in Fig. 11. This publication process determines what objects are visible at particular Web addresses upon discovery instructions.

[0081] At this time a combined object 38 may be instantiated formed of an object provider 110 and a proprietary object 100 (shown in Fig. 10) using the instancing instructions described above. Access instructions from messages 105 are directed to the instanced object 38. The objects 38 may communicate with the communications program 64 and may be stored persistently in storage area 42 as has been described above.

[0082] It will be understood that a program 114 underlying the object provider 110 must be written for each proprietary object 100. However, using the polymorphism of objects much of the software needed for this process may be easily reused. Features not provided by any of the objects 100 or by a particular proprietary object 100 being used may also be implemented by the object provider 110 serving both as a object and as a virtual object for some object methods. In this respect, the object provider 110 may communicate directly with the assenter 108.

[0083] It is specifically intended that the present invention not be limited to the embodiments and illustrations contained herein, but that modified forms of those embodiments including portions of the embodiments and combinations of elements of different embodiments also be included as come within the scope of the following claims.